FOUNDATION-0.10: Shared Utilities - PART 2 (Execution & Validation)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 4 Afternoon)

Component: Shared Python Utilities - Testing & Validation

Estimated Time: 10 min execution

Files: Part 2 of 2 (Execution guide)

MILESTONE: Validate shared utilities with comprehensive testing! ✅

📋 PREREQUISITES

Before starting, ensure you have completed:

- ✅ **PART 1:** All code files generated and reviewed
- ✅ Python 3.11+ installed
- ✅ Base services running (PostgreSQL, Redis, ClickHouse, Qdrant)

🚀 STEP 1: CREATE DIRECTORY STRUCTURE

bash

```
cd ~/optiinfra

# Create shared utilities directories
mkdir -p shared/database
mkdir -p shared/config
mkdir -p shared/logging
mkdir -p shared/utils
mkdir -p shared/examples

# Verify structure
tree shared/ -L 2
# Expected:
# shared/
# ├── database/
# ├── config/
# ├── logging/
# ├── utils/
# ├── examples/
# ├── orchestrator/ (from previous)
# └── mock_cloud/ (from previous)
```

📝 STEP 2: INSTALL ALL FILES FROM PART 1

Copy all 13 files from PART 1:

Database Module

bash

```
cd ~/optiinfra/shared/database
# Copy these files:
# - connections.py
# - __init__.py
```

Config Module

bash

```
cd ~/optiinfra/shared/config
# Copy these files:
# - settings.py
# - __init__.py
```

Logging Module

bash

```
cd ~/optiinfra/shared/logging
# Copy these files:
# - logger.py
# - __init__.py
```

Utils Module

bash

```
cd ~/optiinfra/shared/utils
# Copy these files:
# - retry.py
# - timeseries.py
# - validators.py
# - metrics.py
# - __init__.py
```

Examples and Requirements

bash

```
cd ~/optiinfra/shared
# Copy these files:
# - requirements.txt
# - examples/usage_example.py
```

🔨 STEP 3: INSTALL DEPENDENCIES

bash

```
cd ~/optiinfra/shared

# Install Python dependencies
pip install -r requirements.txt

# Expected output:
# Successfully installed psycopg2-binary-2.9.9
# Successfully installed clickhouse-driver-0.2.6
# Successfully installed qdrant-client-1.7.0
# Successfully installed redis-5.0.1
# Successfully installed python-dotenv-1.0.0
# Successfully installed pydantic-2.5.0
```

Troubleshooting:

```
bash

# If pip install fails, try:
pip install --upgrade pip
pip install -r requirements.txt --no-cache-dir
```

🧪 STEP 4: TEST CONFIGURATION MODULE

```
bash
```

```
cd ~/optiinfra

# Create test script
cat > test_config.py << 'EOF'
#!/usr/bin/env python3
"""Test configuration utilities"""
import sys
sys.path.insert(0, '/home/user/optiinfra')

from shared.config import settings

print("=" * 60)
print("TESTING CONFIGURATION MODULE")
print("=" * 60)

# Test database config
print("\n1. Database Configuration:")
print(f"   PostgreSQL URL: {settings.database.postgres_url}")
print(f"   ClickHouse Host: {settings.database.clickhouse_host}")
print(f"   Qdrant Port: {settings.database.qdrant_port}")
print(f"   Redis DB: {settings.database.redis_db}")

# Test orchestrator config
print("\n2. Orchestrator Configuration:")
print(f"   URL: {settings.orchestrator.url}")

# Test mock cloud config
print("\n3. Mock Cloud Configuration:")
print(f"   URL: {settings.mock_cloud.url}")

# Test logging config
print("\n4. Logging Configuration:")
print(f"   Level: {settings.logging.level}")
print(f"   Format: {settings.logging.format}")

# Test agent config
print("\n5. Agent Configuration:")
print(f"   Name: {settings.agent.name}")
print(f"   Type: {settings.agent.agent_type}")
print(f"   Heartbeat Interval: {settings.agent.heartbeat_interval}s")

# Test environment
print("\n6. Environment:")
```

```
print(f"   Environment: {settings.environment}")
print(f"   Is Production: {settings.is_production}")
print(f"   Is Development: {settings.is_development}")
print(f"   Debug: {settings.debug}")

print("\n" + "=" * 60)
print("✅ CONFIGURATION MODULE TEST PASSED")
print("=" * 60)
EOF

chmod +x test_config.py
python3 test_config.py
```

Expected Output:

```
============================================================
TESTING CONFIGURATION MODULE
============================================================

1. Database Configuration:
   PostgreSQL URL: postgresql://optiinfra:password@localhost:5432/optiinfra
   ClickHouse Host: localhost
   Qdrant Port: 6333
   Redis DB: 0

2. Orchestrator Configuration:
   URL: http://localhost:8080

3. Mock Cloud Configuration:
   URL: http://localhost:5000

4. Logging Configuration:
   Level: INFO
   Format: json

5. Agent Configuration:
   Name: agent
   Type: generic
   Heartbeat Interval: 30s

6. Environment:
   Environment: development
   Is Production: False
   Is Development: True
   Debug: False

============================================================
✅ CONFIGURATION MODULE TEST PASSED
============================================================
```

🧪 STEP 5: TEST LOGGING MODULE

bash

```
cd ~/optiinfra

cat > test_logging.py << 'EOF'
#!/usr/bin/env python3
"""Test logging utilities"""
import sys
sys.path.insert(0, '/home/user/optiinfra')

from shared.logging import setup_logger, log_with_context

print("=" * 60)
print("TESTING LOGGING MODULE")
print("=" * 60)

# Test text logging
print("\n1. Text Logger:")
text_logger = setup_logger('test_text', level='INFO', format_type='text')
text_logger.info("This is an info message")
text_logger.warning("This is a warning")
text_logger.error("This is an error")

# Test JSON logging
print("\n2. JSON Logger:")
json_logger = setup_logger('test_json', level='DEBUG', format_type='json')
json_logger.debug("Debug message with JSON format")
json_logger.info("Info message with JSON format")

# Test context logging
print("\n3. Context Logging:")
log_with_context(
    text_logger,
    'info',
    'Processing request',
    user_id='12345',
    request_id='req-abc',
    duration_ms=125
)

print("\n" + "=" * 60)
print("✅ LOGGING MODULE TEST PASSED")
print("=" * 60)
EOF
```

```
chmod +x test_logging.py
python3 test_logging.py
```

🧪 STEP 6: TEST RETRY DECORATOR

bash

```
cd ~/optiinfra

cat > test_retry.py << 'EOF'
#!/usr/bin/env python3
"""Test retry decorator"""
import sys
sys.path.insert(0, '/home/user/optiinfra')

import random
from shared.utils import retry

print("=" * 60)
print("TESTING RETRY DECORATOR")
print("=" * 60)

attempt_count = 0

@retry(max_attempts=3, delay=0.5, backoff=2.0)
def flaky_function():
    """Function that fails randomly"""
    global attempt_count
    attempt_count += 1

    print(f"\nAttempt {attempt_count}...")

    if random.random() < 0.6:  # 60% failure rate
        raise Exception("Simulated failure")

    return "Success!"

# Test 1: Function that eventually succeeds
print("\n1. Testing retry with eventual success:")
attempt_count = 0
try:
    result = flaky_function()
    print(f"✅ Result: {result} (after {attempt_count} attempts)")
except Exception as e:
    print(f"❌ Failed after {attempt_count} attempts: {e}")

# Test 2: Function that always fails
print("\n2. Testing retry with persistent failure:")
@retry(max_attempts=3, delay=0.2, backoff=1.5)
def always_fails():
```

```
    raise ValueError("This always fails")

try:
    always_fails()
except ValueError as e:
    print(f"✅ Correctly failed after retries: {e}")

print("\n" + "=" * 60)
print("✅ RETRY DECORATOR TEST PASSED")
print("=" * 60)
EOF

chmod +x test_retry.py
python3 test_retry.py
```

🧪 STEP 7: TEST TIME SERIES HELPERS

bash

```
cd ~/optiinfra

cat > test_timeseries.py << 'EOF'
#!/usr/bin/env python3
"""Test time series utilities"""
import sys
sys.path.insert(0, '/home/user/optiinfra')

from datetime import datetime, timedelta
from shared.utils import (
    create_time_windows,
    aggregate_time_series,
    calculate_rate_of_change,
    detect_anomalies,
)

print("=" * 60)
print("TESTING TIME SERIES MODULE")
print("=" * 60)

# Test 1: Create time windows
print("\n1. Creating time windows:")
start = datetime(2025, 1, 1, 0, 0)
end = datetime(2025, 1, 1, 6, 0)
windows = create_time_windows(start, end, timedelta(hours=1))
print(f"   Created {len(windows)} hourly windows")
print(f"   First window: {windows[0]}")
print(f"   Last window: {windows[-1]}")

# Test 2: Aggregate time series
print("\n2. Aggregating time series data:")
data = [
    {'timestamp': datetime(2025, 1, 1, 0, 5), 'cpu': 45.2},
    {'timestamp': datetime(2025, 1, 1, 0, 15), 'cpu': 52.1},
    {'timestamp': datetime(2025, 1, 1, 0, 25), 'cpu': 48.5},
    {'timestamp': datetime(2025, 1, 1, 0, 35), 'cpu': 51.0},
    {'timestamp': datetime(2025, 1, 1, 1, 5), 'cpu': 47.3},
    {'timestamp': datetime(2025, 1, 1, 1, 15), 'cpu': 49.8},
]

aggregated = aggregate_time_series(
    data=data,
    timestamp_field='timestamp',
```

```
    value_field='cpu',
    window_size=timedelta(hours=1),
    aggregation='avg'
)

print(f"   Aggregated {len(data)} points into {len(aggregated)} windows")
for agg in aggregated:
    print(f"   Window {agg['window_start'].strftime('%H:%M')}: avg={agg['value']:.2f}, count={agg['count']}")

# Test 3: Calculate rate of change
print("\n3. Calculating rate of change:")
rate_data = calculate_rate_of_change(
    data=data,
    timestamp_field='timestamp',
    value_field='cpu'
)
print(f"   Calculated rate for {len(rate_data)} points")
print(f"   Sample: {rate_data[0]}")

# Test 4: Detect anomalies
print("\n4. Detecting anomalies:")
values = [10, 12, 11, 10, 50, 11, 12, 9, 45, 10]  # 50 and 45 are anomalies
anomaly_indices = detect_anomalies(values, threshold_std_dev=2.0)
print(f"   Detected {len(anomaly_indices)} anomalies at indices: {anomaly_indices}")
print(f"   Anomalous values: {[values[i] for i in anomaly_indices]}")

print("\n" + "=" * 60)
print("✅ TIME SERIES MODULE TEST PASSED")
print("=" * 60)
EOF

chmod +x test_timeseries.py
python3 test_timeseries.py
```

🧪 STEP